What's new for virtual system patterns in Pure Application System 2.0

Kirti Apte (kaapte@us.ibm.com)
Software Engineer, IBM PureApplication System
development team
IBM

Qi Liu
Software Engineer, IBM PureApplication System
development team
IBM

Shu Chao Wan (wansc@cn.ibm.com)
Advisory Software Engineer, IBM PureApplication System
development team
IBM

Yue Wang (ywangwy@cn.ibm.com)
Advisory Software Engineer, IBM PureApplication System
development team
IBM

IBM PureApplication System 2.0 introduces a new style of virtual system patterns with many enhancements and improved lifecycle management functionality. This tutorial gives you the details.

Introduction

IBM® PureApplication® System is an expert-integrated system offering that provides integrated computing, networking, storage, hardware management, and middleware deployment. The middleware deployment uses patterns, which are templates for multi-tier enterprise solutions. The pattern engine is an integral part of PureApplication System, and it provides deployment and lifecycle management for patterns. PureApplication System supports two kinds of pattern models:

Virtual system pattern

A topology-centric definition of a multi-tier middleware composition. A virtual system pattern might include a WebSphere Application Server Network Deployment topology, an IBM HTTP Server front end, a database such as IBM DB2, a user registry such as LDAP, and a
messaging engine such as IBM MQ. A virtual system pattern defines topology configuration through virtual images and automation through script packages.

**Virtual application pattern**
An application and service-level-agreement (SLA) centric definition that includes application components, such as Java Platform Enterprise Edition (JEE) application archives like EAR and WAR files, and a corresponding SLA, such as a response time less than three seconds. PureApplication System 2.0 introduces a new style of virtual system patterns that are the focus of this article. Virtual system patterns that were built before PureApplication System 2.0 are now referred to as *classic virtual system patterns*, and they are still fully supported for deployment and management. The new virtual system patterns, however, provide many enhancements to offer richer lifecycle management functionality. Virtual application patterns remain unchanged and are fully supported.

**Building blocks for virtual system patterns**
Classic virtual system patterns use three building blocks:

**Hypervisor images**
Virtual images built for a specific hypervisor, such as VMware or PowerVM. They contain an operating system image, middleware, and pre-defined scripts that configure the middleware and operating system when the image is instantiated. For example, a WebSphere Application Server Hypervisor image for the Red Hat Linux (RHEL) Operating System for VMware is a VMware-specific virtual image that contains the RHEL operating system, WebSphere Application Server installation binary files for a specific version and fix pack, and scripts that configure WebSphere Application Server as a standalone application server or deployment manager.

**Script packages**
Compressed files in ZIP format that contain arbitrary automation that you can orchestrate as part of pattern lifecycle. These packages can be run at virtual machine (VM) instantiation, VM shutdown, or any time in between in a manual mode.

**Add-ons**
Special script packages that handle operating system security, storage, or networking configuration.

While IBM and its IBM Business Partner ecosystem provide virtual images for various products to be used in virtual system patterns, it is common for clients to customize these images to make them compliant in their environments. These customizations commonly include changing the operating system configuration, hardening security, installing IBM or third-party software or agents, and so on, to meet security and data center compliance requirements. Consider that for every hypervisor image that is shipped from a vendor (for every release and fix pack) clients typically produce another set of those images that are compliant with their corporate standards, thus doubling the number of images to manage, and often creating "image sprawl."

Secondly, a group of administrators often manages operating system compliance, while another group manages middleware configuration. Packaging the operating system and middleware in a single image requires two sets of administrators to work on a single asset, and prevents a clear separation of duties.
Thirdly, packaging middleware with the operating system requires extra work to compose an additional middleware stack within a single image instance -- for example, having WebSphere Application Server and DB2 in a single virtual machine for a development environment, as opposed to deploying a WebSphere Application Server virtual machine and a DB2 virtual machine.

To address these issues with hypervisor image-based classic virtual system patterns, the new virtual system patterns (sometimes referred to as vsys.next), separate the operating system from the middleware and allow dynamic composition of middleware on top of a specified base operating system image, as shown in Figure 1:

**Figure 1. The operating system is now separated from the middleware**

A *software component* is a middleware product with lifecycle management automation (scripts) that can then be deployed and managed by the PureApplication System pattern engine. Separating software components from the operating system image has many advantages:

- A single operating system image can be customized to corporate standards and then used with various middleware components, thus reducing image sprawl.
- The corresponding operating system image can be small by default, and sized appropriately for the middleware.
- Multiple software components can be stacked in a single operating system instance.
- Software components can be reused. Fix packs can declare dependencies on base release contents. Software components can declare dependencies on other software components to build stack products.
- Operating system maintenance can be decoupled from the middleware lifecycle.
- Software components are easily portable across operating system versions, hypervisors, and other platforms.

The other two building blocks in classic virtual system patterns -- script packages and add-ons -- are still used in virtual system patterns. However, there are multiple enhancements to the pattern building experience and deployment/runtime features.

You can now take an existing classic virtual system pattern and promote it to a virtual system pattern so it can use the new pattern building and runtime features offered by the virtual system deployment model. A promoted virtual system pattern is decoupled from the original classic virtual system pattern, and therefore, you can manage and deploy each one independently.

Virtual system (vsys.next) patterns support version tags associated with the pattern. PureApplication System does not offer a versioned repository. Therefore, treat the version only as a tag used to group pattern definitions or identify pattern evolution. PureApplication System 2.0 also supports versions for script packages and add-ons.
Pattern creators use the Pattern Builder to build virtual system patterns -- the same tool that is used to build virtual application patterns. As shown in Figure 2, the Pattern Builder includes a palette on the left that enumerates images, software components, and script packages from which you can build virtual system patterns. The hypervisor images from classic virtual system patterns are also represented as images in the Pattern Builder, along with the base operating system image, and can be used in composing virtual system patterns.

**Figure 2. New options in the Pattern Builder**

You can drag images onto the canvas and compose the software stack within the image by dragging software components and script packages into the image, as shown in Figure 3:

**Figure 3. Creating a software stack with software components and script packages**
Add-ons are now treated similar to policies in the virtual application builder and are available on the image toolbar, as shown in Figure 4 below. PureApplication System 2.0 provides detachable block storage among other artifacts, which displays as an add-on for use by patterns.

**Figure 4. Add-ons are available from the image toolbar**

![Image showing add-ons available from the image toolbar](image-url)

**Policy association**

A virtual system pattern, similar to a virtual application pattern, allows policy association to artifacts defined within the pattern. Policies enhance the runtime behavior of the deployed environment. The following policies are implemented out-of-box:

**Security policy**
Enables/disables password-based SSH login to the virtual machines.

**GPFS Client policy**
Configures the IBM General Parallel File System (GPFS) client software and mounts a shared GPFS file set to the virtual machine.

**Interim fix policy**
Accepts a list of emergency fixes that can be applied to this virtual machine during deployment. The fixes must be added to the catalog and associated with the image or software components.

**Placement policy**
Specifies placement constraints for like virtual machines for collocation or anti-collocation. For example, members of a cluster should be anti-collocated.

**Routing policy**
Allows an endpoint that is defined within a specific virtual machine to be routed from an elastic load balancer.

**Base Scaling policy**
Provides multiple instances of a virtual image and software stack within it to be instantiated at deployment time, and then enables scaling of these instances (in or out) based on processor or memory consumption.

Not all policies are applicable in each environment. While the base operating system image provided with PureApplication System System supports these policies, the software components that are added to these images during pattern definition might be configured to not support a
certain policy or portion of a policy (such as horizontal scaling). In these cases, the corresponding policy or portion of the policy is removed from the list of applicable policies that is shown for the image.

Policies can be applied globally at the pattern level or specified for individual images. Pattern-level policies are applied to the entire pattern and can be added as shown in Figure 5:

**Figure 5. Several pattern-level policies are available**

![Pattern-level policies](image)

Individual image-level policies are applied for a specific virtual machine, as shown in Figure 6:

**Figure 6. Several image-level policies are available**

![Image-level policies](image)

**Data dependencies**

Within a virtual system pattern, images, scripts, software components, and add-ons have attributes whose values are entered by the pattern creator or pattern deployer. Often, the value of a particular attribute is either not known at pattern creation time, or else depends on the value provided for another attribute. For example, a script package to configure a data source on an application server virtual machine requires the host name and port number of the virtual machine that runs the database, and also the name of the database that is created by another script package on the database virtual machine. The host name or IP address are not known until the
pattern is deployed. Also, the name of the database can be entered by the pattern creator in the database creation script, and the exact same name must also be entered in the data source configuration scripts. Such attribute dependencies are specified by data dependency links.

Each artifact within a virtual system pattern (for example, image, software component, or script package) has a set of outputs that are defined in its metadata, and a set of attributes (inputs) whose values are entered by the pattern creator or deployer. An image also has default attributes, such as host name and IP address. A data dependency link between attributes of two components specifies that the value of an attribute of one component depends on (or is same as) the value of an attribute of another component. Figure 7 shows a `Sample_TradeLite_CreateDataSource` script within the WebSphere Application Server node that has attributes such as `DATABASE_NAME`, `DATABASE_USERNAME`, and `DATABASE_HOST`. Their values depend on (or need to match) the corresponding attributes in the `CreateDB_OLTP` script that is added to DB2 node:

**Figure 7. Data dependency links**

A data dependency link is drawn from the component whose attribute depends on another attribute. The source of the link is the attribute that needs the value and the target of the link is the attribute that supplies the value.

**Pattern-level attributes**

When you build complex patterns, you often have an attribute across a set of artifacts for which a deployer typically provides the same value during pattern deployment. A common example is a password for users that is created as part of pattern deployment. Classic virtual system patterns let you enter the operating system user password only once and apply it to all virtual machine instances during deployment. Virtual system patterns generalize this concept by using **pattern-level attributes**.

A pattern-level attribute is a user-defined attribute that is associated with the entire pattern. Any artifact, such as an image, a script package, an add-on, or a software component, can be linked to this pattern-level attribute. This linkage means that the value specified for the pattern-level attribute is propagated to all of the attributes that are linked to it. Therefore, you can create a password
attribute at the pattern level and link all of the operating system user passwords for all virtual machines to it.

Similarly, you can choose any other attribute, such as a thread pool value within various script packages, or an installation directory for a specific software component, across all virtual images to be defined as a pattern-level attribute, and then link it to individual attributes. This linkage makes attribute specification less error-prone by consolidating the same attributes under a single definition. Figure 8 shows an example of the pattern-level attribute definition for the WebSphere Application Server administrative user name attribute:

**Figure 8. Pattern-level attribute definition**

![Pattern-level Attribute Definition](image)

Figure 9 shows the reference to this attribute in the WebSphere Application Server software component that is embedded in the pattern:
Ordering

Unlike classic virtual system patterns, the deployment process for virtual system patterns is highly parallelized for faster deployment times. All virtual machines are deployed in parallel. Additionally, the software stack within each virtual machine is deployed in parallel as much as possible. Within the Pattern Builder, the behavior of this execution is controlled by the ordering view tab, which shows *swimlanes* for each of the virtual machines. Each swimlane is executed in parallel, and by default, each virtual machine is in its own swimlane.

You can create ordering constraints between components across swimlanes. Data dependency that is defined between components automatically creates an ordering dependency. During pattern deployment, each swimlane is executed in parallel until an ordering constraint is hit, and then the ordering constraint is executed sequentially. After the sequential execution finishes, the deployment processing resumes with parallel execution. Figure 10 shows swim lanes for a pattern that contains multiple virtual machines:

Integration with Installation Manager

Cloud administrators can manage Installation Manager repositories. When cloud administrators import pattern types into PureApplication System, the Installation Manager-based software
packages are automatically uploaded to storehouse and the internal Installation Manager repository. Cloud administrators can also manually add the fixes or software packages to the internal Installation Manager repository, or configure a connection to an external Installation Manager repository.

**Adding content to the internal Installation Manager repository**

Cloud administrators can manually add software packages to the repository by using these steps:

1. Click **Catalog => IBM Installation Manager Repository**.
2. Click **Add New**, and then:
   1. Click **Add New Category** if there is no existing category for the software package.
   2. Click **Add New Software Package** if an existing category is suitable for the software package.
   3. Browse to the compressed file for the software package.
   4. Set the category.
   5. Click **Add**.

**Figure 11. Add content to the internal Installation Manager repository**

![Add New Software Package](image)

**Configure a connection to an external Installation Manager repository**

Cloud administrators can use the following steps to configure a connection to an external Installation Manager repository, so that they can use software packages from the repository in virtual system patterns:

1. Click **Cloud => System Plug-ins**.
2. Select the **Virtual System Pattern Type**.
3. Select the **vsys.im** plug-in.
4. Click **Configure**.
   1. Specify the **External Repository URL**.
   2. Specify the **User Id** and **Password** for the repository user.
   3. Click **Update**.
5. Optional: Click **Test Connection** to confirm that the connection to the repository works.
Figure 12. Configure the connection to an external Installation Manager repository

Deploying virtual system patterns

Deploying a virtual system pattern follows the same process as a virtual application pattern. The deployment dialog, which is shared by both types of patterns, has been improved in PureApplication System 2.0 to make it simpler and more intuitive.

After you save a newly created virtual system pattern, you can view it by clicking Patterns => Virtual Systems. In addition to the patterns that you created, you can see some built-in applications that are installed with the default pattern types. One of these patterns is the BPM Process Center with Embedded DB2 virtual system pattern, a complex pattern shipped with the IBM BPM Pattern 8.5.5 for Virtual System 1.0.0.0 pattern type. If you open the pattern, you can see its topology:

Figure 13. Topology for the BPM Process Center with Embedded DB2 virtual system pattern

This pattern contains an IBM BPM custom node software component with a scaling policy attached that is federated to a BPM Process Center Deployment Manager software component. The Process Center IHS is regarded as a load balancer and an entry point to dispatch requests to custom nodes. The Primary DB2 and Standby DB2 work together to support high availability and disaster recovery (HADR). All of these software components are installed on a Red Hat Linux
operating system image, and multiple script packages and add-ons are attached to the software components. Select the pattern, and then click **Deploy** to open the deployment dialog in another web browser tab.

In the configuration panel of this new deployment dialog, you can specify the deployment target and options in the left pane, including the deployment’s name, environment profile, cloud group, IP group, and other configuration options. In the right pane, you can configure all of the locked attributes in the pattern, including pattern-level attributes and component-level attributes. The deployment process can be continued only after you configure all of the required information. You can click **Quick Deploy** on this page to start the deployment immediately, in which case the distribution and network setting are determined by system, or you can click **Continue to Distribute** to open the distribution page, where you can configure the distribution and network settings. After you configure the settings on the distribution page, click **Deploy** to deploy the pattern. For details about the options on this dialog, see [Deploying virtual system patterns](#) in the PureApplication System Knowledge Center.

**Figure 14. New deployment dialog**

After deployment, a virtual system instance is generated. The virtual system instance in PureApplication System 2.0 contains not only the capability of a classic virtual system instance, for example, processor, memory metering, script package execution and logging, and console links, but also new functions, which are derived from virtual application instances, such as logging, monitoring, and operations. On the **Instances** page, which you can open by clicking **Instances => Virtual Systems**, you can find your deployment by searching for its name. In the detail pane for the instance, you can see its status, environment profile, cloud group, pattern type, pattern, all virtual machine details, history, and other details. You can also start, stop, maintain, resume, and delete the instance, and also create and restore snapshots. Open the **Instance Console** to see more available management actions for the instance, such as logs, execute operations, and monitoring for both the virtual machines and middleware use.
On the Instances page, you can also see detailed information for each virtual machine in the instance, such as IP address, running status, hardware, network, and operating system information. The management actions for a particular virtual machine, including start, stop, execute script packages, download script package execution log files, and access console links, are also available from this page, as shown in Figure 16:

**Developing Software Components**

PureApplication System provides both an Eclipse-based Plug-in Development Kit (version 1.1), and a web tool to construct software components. The Plug-in Development Kit (PDK) includes editors and utilities to build software components. PureApplication System uses plug-ins to define software components, so the project structure and configuration files are similar to a plug-in project in a virtual application pattern in first compression, but with more attributes to support new capabilities. Definitions are also simplified; for example, the transformer is no longer required.
Use the following steps to create a sample software component that includes the new required properties.

**Software component creation using Eclipse-based PDK**

1. Create a project skeleton with the software component wizard. In Eclipse, select **File => New => Project => IBM Workload Software Component Project**, and then click **Next** to start the wizard.
2. Specify the project name and location (if it is not the default location).
3. Click **Finish**, or if you need to configure the primary pattern type, click **Next** and specify a primary pattern type.
   These steps are same as the creation of virtual application pattern plug-in project. The PDK generates the software component project skeleton.
   The overview editor is opened after the PDK creates the software component project. The contents of the newly created project are almost the same as a virtual application pattern plug-in project. Additional configuration settings include setting the virtual system pattern type as a secondary pattern type, and the overview file in the project root for reopening the overview editor. In PDK v1.1, if you click a single JSON configuration file in a software component project, the PDK opens only the source editor instead of opening form editors and source editors for all of the configuration files, as in a virtual application pattern plug-in project.
4. Configure the packages for the software component. In the overview editor, switch to the **Packages** tab. In the **Basic Information** part, you can configure:
   - **Name**
     Plug-in name.
   - **Version**
     Plug-in version.
   - **Pattern types**
     You can define only one primary pattern type and multiple secondary pattern types. vsys is required, and must be included in the pattern types. By default, it is set as secondary pattern type.
   - **Artifacts**
     Specify the relative path that is used in storehouse for the installation files or other artifacts that must be packaged with the plug-in (primary pattern type). Lifecycle scripts in the target virtual machine can download these artifacts.
   - **Plug-in parameters**
     Plug-in parameters, which are listed on the plug-in details page.
   - **IM artifacts**
     IM artifacts are a new feature in software components. Use this field to define artifacts to import into the internal Installation Manager repository with this plug-in. Click **IM Artifacts** to see a table of Installation Manager Artifacts for the project. You can add, edit, and remove Installation Manager Artifacts for the plug-in by using this dialog. Click **Add** to open the add dialog and add an artifact. The relative path that is referenced by the **Add** page is the path that is used to store the artifact in storehouse, similar to the **Relative path** field on the **Artifacts** page. For the Installation Manager repository, this path is used to store artifacts in the internal Installation Manager repository before installation. The **Category** is used to specify the category for the artifact in the Installation Manager repository.
Properties that are marked with an asterisk are required.

5. Configure the **Package** section. In the Package section, you can create a package, remove a package, and add support for multiple operating systems in one package. You can also combine support for multiple operating systems into one support element. This configuration depends on the granularity of the script for each platform or operating system. The Package section contains these configurable parameters:

**Package name**
The package identifier.

**Architecture/OS/OS version**
Specify the supported platforms and operating systems for scripts in this package. You can select or input values.

**CPU/storage/Memory**
Specify additional resources for product execution or installation (storage). The system will validate the storage with selected operating system image during deployment.

**Dependent products/Package provides**
Specify dependent projects and projects that are provided in this package.

**Included parts**
Specify the part members for the package. Candidate parts are defined in the plugin/nodeparts and plugin/parts folders in the project, which include scripts that can be executed in the deployed virtual machine (VM). Creation of new node parts or parts is the same as for virtual application pattern plug-ins. Select **New => Plug-in Part** (or **Plug-in Node Part**) to start the wizard and select the part level script files. If you create a part, you must create a role in that part. Use the **Plug-in Role** wizard to create a role, and then add this part or node part to the package.

6. Select the **Components** tab to configure the components for the software component.

In components list part, click **Add** to create a new component. Software components are displayed in the Pattern Builder palette. Pattern creators can drag the components onto the canvas to add them to the virtual pattern. Configure the attributes for the component:

**ID**
The component identifier.

**Label**
The label that is shown in the component list or on the Pattern Builder canvas. It can be globalized.

**Extends**
Specify the parent component for this component to inherit. Part is the root component for all components. The default value is Part. You can specify other components in this plug-in or other related plug-ins as the parent.

**Images/Thumbnail**
Specify the images that are used to represent this component in the palette list, pattern builder, pattern details page, and other pages where it is referenced. The required sizes are 64X64 and 48X48 pixels.

**Attributes**
When a pattern creator selects a component in the Pattern Builder, the pattern creator can update these attributes in the Pattern Builder. These attributes can also be updated during
pattern deployment. These attributes can be accessed in lifecycle scripts. This section is the same as for a virtual application plug-in.

**Outputs**
This attribute is new to software components. It is used to define attributes that this component wants to make available to other components. When you connect two components in the Pattern Builder, an output in the source component can map to attributes in the target component by configuring mappings in the Pattern Builder. Usually, each output binds to an attribute. When you define an attribute, you can click **Exposé as output** to add it to outputs.

**Related package**
Define related packages, which include scripts to run or features that are required for this package. This parameter has three tabs:

**Package**
Specify the primary package for this component. The part key is the name of plug-in role in this package for execution. The part key is a single value, so if multiple roles are defined in these packages, other roles will not run. For each **Support** element in a package, included parts are required to include the same role as the part key.

**Install via IM**
Specify whether the Installation Manager repository is required for installation. In the **IM offering ID filter**, specify a keyword to help the system find the artifact in the repository, which might contain hundreds of elements.

**YUM packages**
Define additional packages that the product requires but that the base image does not include.

**Console links**
Define links that are provided by installed products that are shown on the deployed instance details page. There are some system variables can be used in URL definition:

- public-hostname: for example, https://{public-hostname}/vnc
- public-ip: for example, https://{public-ip}/vnc

7. Define operations that an administrator can run after the pattern is deployed on the runtime operation tab. This page is the same as a virtual application pattern plug-in.

After you complete the configuration and develop the lifecycle scripts, the PDK can build and install the plug-in projects to your test environment. The PDK also provides a runtime perspective to retrieve logs from deployed virtual machines to help debug issues. These functions are also available for virtual application pattern plug-in projects. For more information, see [Building a project](#) and [Debug plug-in](#)

**Creating software components by using the workload console**

You can also create software components from the workload console.

1. Select menu **Catalog => Components Definition** from the PureApplication System workload console.
2. Click **New** in the toolbar.
3. Select **Create a Software Component**, and click **Next**.
4. Specify the Name, Version, Label, Description, and Operating system attributes, which are the same as in Packages tab that was described in the previous section.

5. Select icon and thumbnail images.

6. Click Create to display the details definition page for the software component. You can configure the following attributes on the details page:
   • Basic information, including the Name, Version, Label, Operating System, Extends, Description, Icon image, and Thumbnail image, which you set on the previous page.
   • Properties and Outputs: These attributes are the same as the attributes that were described on the Components tab in the previous section.
   • Lifecycle Operations: These operations are the same as the lifecycle scripts that are in the /plugin/parts/[part name included in required package]/scripts/[part key]/folder of an Eclipse-based project. Each lifecycle script (install/configure/start/stop) can map to only one python script, but other script files can be packaged as artifacts and be started in a mapped script.
   • Artifacts: This section is the same as the Artifacts section of Packages tab that was described in the previous section. One simplification in the web-based tool is that you can define a Deploy path, and the system generates a script to download artifacts to this path, which can be accessed by lifecycle scripts.
   • Console links: This section is the same as the Console links section that was described for the Components tab in the previous section.
   • After you complete the configuration, click Publish on the toolbar to install this software component to PureApplication System and make it available in the Pattern Builder palette.

Conclusion

This article introduced the new style of virtual system patterns that were introduced in PureApplication System 2.0 and the many enhancements that this new style provides. You learned how to build virtual system patterns, including how to use and build software components. You also learned how to deploy the virtual system patterns and use the features that are available to manage those deployed instances.

Acknowledgments

The authors would like to thank Ajay Apte, STSM - PureApplication System and IBM Workload Deployer, and Mark Yi, STSM - PureApplication System and IBM Workload Deployer, for their help in reviewing this article.
Resources

- **PureApplication System resources**
  - IBM PureApplication System W2500 V2.0 documentation
  - IBM PureApplication System W2700 V2.0 documentation
  - Installing the Plug-in Development Kit
    This topic in the PureApplication System documentation provides instructions to download and install the Plug-in Development Kit (PDK)

- **WebSphere resources**
  - developerWorks WebSphere
    Technical information and resources for developers who use WebSphere products. developerWorks WebSphere provides downloads, how-to information, support resources, and a free technical library of more than 2000 technical articles, tutorials, best practices, IBM Redbooks, and product manuals.
  - developerWorks WebSphere application integration developer resources
    How-to articles, downloads, tutorials, education, product info, and other resources to help you build WebSphere application integration and business integration solutions.
  - Most popular WebSphere trial downloads
    No-charge trial downloads for key WebSphere products.
  - WebSphere forums
    Product-specific forums where you can get answers to your technical questions and share your expertise with other WebSphere users.
  - WebSphere demos
    Download and watch these self-running demos, and learn how WebSphere products can provide business advantage for your company.
  - WebSphere-related articles on developerWorks
    Over 3000 edited and categorized articles on WebSphere and related technologies by top practitioners and consultants inside and outside IBM. Search for what you need.
  - developerWorks WebSphere weekly newsletter
    The developerWorks newsletter gives you the latest articles and information only on those topics that interest you. In addition to WebSphere, you can select from Java, Linux, Open source, Rational, SOA, Web services, and other topics. Subscribe now and design your custom mailing.
  - WebSphere-related books from IBM Press
    Convenient online ordering through Barnes & Noble.
  - WebSphere-related events
    Conferences, trade shows, Webcasts, and other events around the world of interest to WebSphere developers.

- **developerWorks resources**
  - Trial downloads for IBM software products
    No-charge trial downloads for selected IBM® DB2®, Lotus®, Rational®, Tivoli®, and WebSphere® products.
  - developerWorks business process management developer resources
    BPM how-to articles, downloads, tutorials, education, product info, and other resources to help you model, assemble, deploy, and manage business processes.
• **developerWorks blogs**
  Join a conversation with developerWorks users and authors, and IBM editors and developers.

• **developerWorks tech briefings**
  Free technical sessions by IBM experts to accelerate your learning curve and help you succeed in your most challenging software projects. Sessions range from one-hour virtual briefings to half-day and full-day live sessions in cities worldwide.

• **developerWorks podcasts**
  Listen to interesting and offbeat interviews and discussions with software innovators.

• **developerWorks on Twitter**
  Check out recent Twitter messages and URLs.

• **IBM Education Assistant**
  A collection of multimedia educational modules that will help you better understand IBM software products and use them more effectively to meet your business requirements.
About the authors

Kirti Apte
Kirti Apte is a Software Engineer on the IBM PureApplication System development team at the IBM Software Lab in Austin, TX, USA. You can contact Kirti at kaapte@us.ibm.com.

Qi Liu
Qi Liu is a Software Engineer on IBM Pattern Development Kit (PDK) team at the IBM China Software Development Lab. Before working on the PDK, he worked on the Composite Application Pattern (CAP) project. You can contact Qi at lqliulq@cn.ibm.com.

Shu Chao Wan
Shu Chao Wan is a Software Engineer on the IBM PureApplication System development team at the IBM China Software Development Lab. She works on the PureApplication System user interface, and her areas of interest include cloud computing and Web 2.0. You can contact Shu Chao at wansc@cn.ibm.com.

Yue Wang
Yue Wang is an Advisory Software Engineer on the IBM PureApplication System development team at the IBM China Software Development Lab. Previously, she worked on the WebSphere Service Registry and Repository team, and she helped deliver industry solutions in the banking, education, healthcare, and telecom industries. Her areas of expertise include cloud, SOA, and industry solutions. You can contact Yue at ywangwy@cn.ibm.com.